Unidad II: Programación orientada a objetos básica

Lic. Ronaldo Armando Canizales Turcios

Departamento de Electrónica e Informática, UCA

Ciclo virtual 01-2021

Agenda

1. Repaso: Abstract
2. Interfaces - teoría
3. Herencia vs interfaces
4. Ejemplo: tipos de vendedores
5. Implementando múltiples interfaces

1. Repaso: Abstract

Palabra reservada abstract - forzando la herencia

Permite crear clases y métodos que están **incompletos** y deben ser implementados en una clase derivada (sub-clase).

Características de las clases abstractas

- Una clase abstracta NO puede ser instanciada (crear objetos).
- Puede contener métodos abstractos.
- Una clase B (no abstracta) que herede de una clase A (abstracta) debe proveer implementaciones de todos los métodos abstractos que ha heredado.
- Una forma de verlos es que permite crear "plantillas" que se utilizarán en las clases hijas.

1. Repaso: Abstract

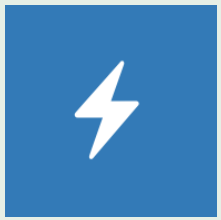
Características de los métodos abstractos

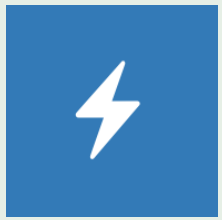
- Sólo pueden ser declarados dentro de una clase abstracta.
- Los métodos abstractos no proveen implementación, por ello no tienen cuerpo (no tienen llaves de apertura o cierre, ni instrucciones dentro) sin embargo, deben terminar con un punto y coma.

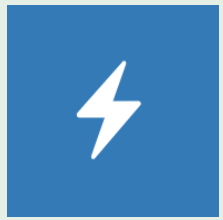
  ¿Dónde se digita su implementación? Debe sobrescribirse dicho método en una clase no abstracta que lo herede (en una sub-clase).

Sintaxis de método abstracto

public abstract void nombreMetodo();


Ejemplo 1


Ejemplo 2


Ejemplo 3

2. Interfaces - teoría

Una interfaz es similar a una clase, con la diferencia de que la interfaz provee una **especificación** en lugar de una **implementación** para sus miembros (métodos). Las interfaces son especiales en los siguientes sentidos:

- Los miembros de una interfaz son todos implícitamente abstractos. En contraste, una clase puede proveer ambos, métodos tanto abstractos como concretos (con su implementación).
- Una clase puede implementar **múltiples** interfaces. En contraste, una clase puede heredar solamente de una única clase padre.

La declaración de una interfaz es parecida a la declaración de una clase, pero no se provee implementación para sus miembros (esto porque todos sus miembros son implícitamente abstractos). Dichos métodos pueden ser implementados por las clases que implementen dicha interfaz.

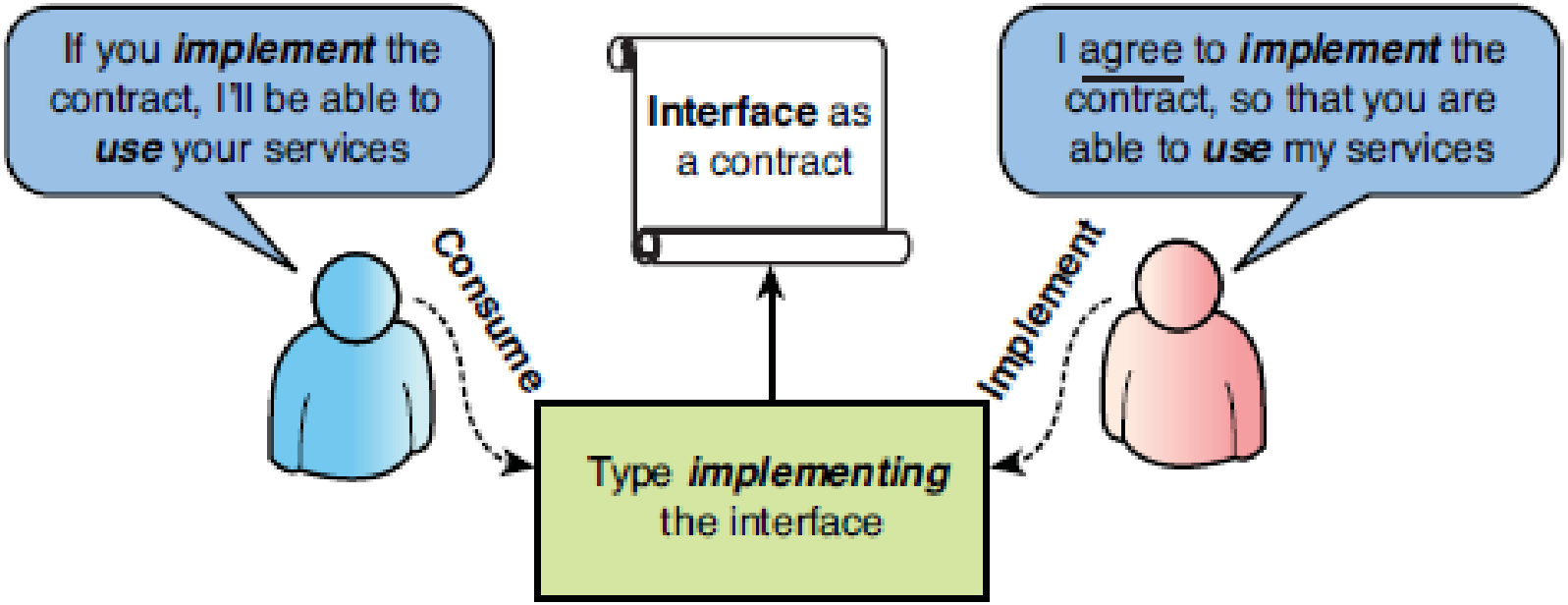
2. Interfaces - teoría

Podemos imaginar las interfaces como “cláusulas en un contrato”, por poner un ejemplo supongamos que contratamos a X compañía telefónica, nosotros les pagaremos por hacer lo siguiente:

- Proveer Internet a nuestras casas.
- Darnos “n” canales de cable.
- Proporcionar un número telefónico exclusivo para nuestro uso.

Pero, ¿realmente nos interesa cómo hacen el papeleo dentro de su empresa, el tamaño de sus oficinas, el color de su logo, el apellido del dueño, cuántos departamentos tienen? es decir **¿importa la implementación específica?**

2. Interfaces - teoría



No, la verdad no, con tal que cumplan lo que han prometido es suficiente. **Cada empresa telefónica hará su papeleo de una forma diferente, pero todas deben cumplir lo que han firmado en sus contratos.**

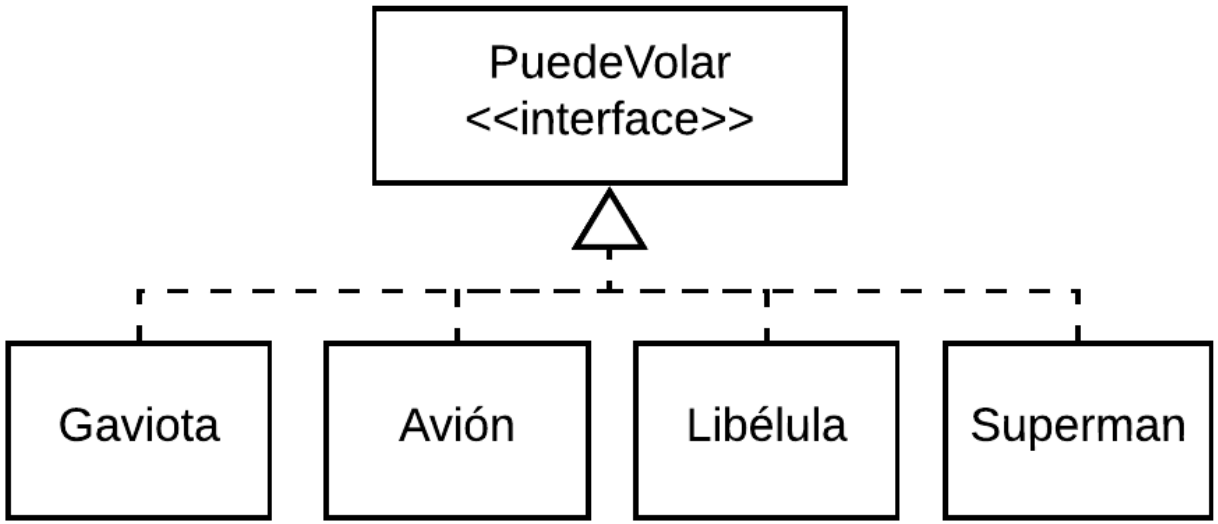
3. Herencia vs interfaces

Un punto de referencia puede ser:

- Usar **herencia** cuando naturalmente (sin forzar) se tiene una implementación compartida. Ejemplo: cuando una clase es un caso específico de otra, como CuentaBancaria y CuentaChequera.
- Usar **interfaces** cuando las implementaciones son independientes. Ejemplo: se busca el mismo objetivo pero el proceso para realizarlo se logra de **maneras distintas**, como la acción de **volar**.




3. Herencia vs interfaces



Código fuente
Versión 1

UML

La implementación de una interfaz se representa con una línea discontinua que nace en la clase y termina en una flecha blanca apuntando a la interfaz. Se diferencia de la herencia (que también tiene una flecha blanca) en la línea que los une, que es continua.

3. Herencia vs interfaces

Polimorfismo e interfaces

De manera análoga a como hicimos variables polimórficas con la herencia, se puede hacer lo mismo con interfaces.

Código fuente
Versión 2

¿Por que? ¿Tiene lógica esto?

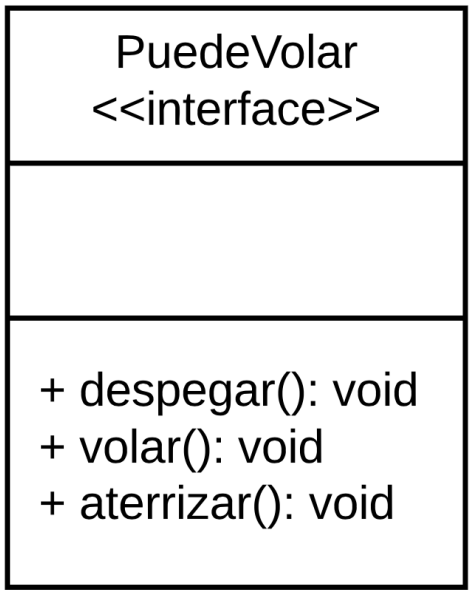
Sí, después de todo... todas las entidades que "PuedanVolar" son capaces de ejecutar los métodos que se listan en dicha interfaz (aunque cada una de forma diferente).

3. Herencia vs interfaces

En la declaración de una interfaz se listan todos los métodos que **requiere** dicha interfaz. Ejemplo: **para que algo pueda volar necesita al menos tres cosas**: despegar, volar y aterrizar.

Sintaxis de una interfaz

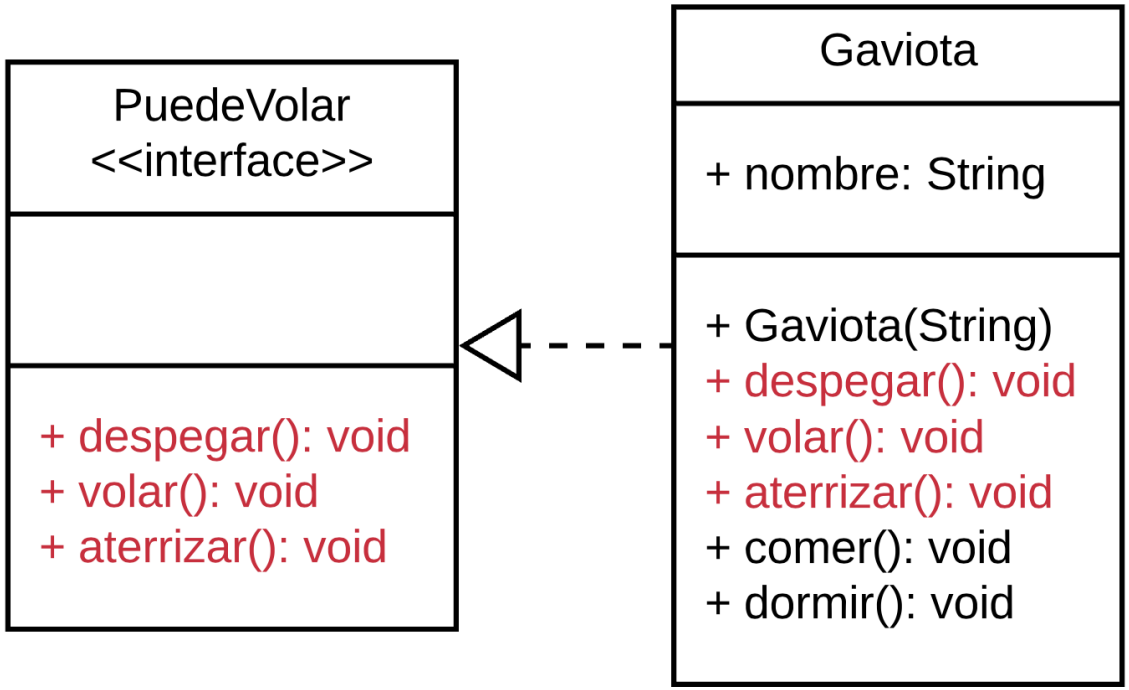
```
public interface PuedeVolar {
    void despegar();
    void volar();
    void aterrizar();
}
```



Los métodos en una interfaz son abstractos, es decir, tienen un nombre, lista de parámetros y tipo de retorno, pero no tienen implementación.

3. Herencia vs interfaces

Dichos tres métodos son suficientes para que una Gaviota pueda volar, pero aparte también pueden tener otros métodos: dormir, comer, etc.



Una interfaz puede requerir que exista uno, dos, tres, diez. . . cualquier cantidad de métodos. Los métodos de una interfaz **deben ser públicos**.

4. Ejemplo: tipos de vendedores

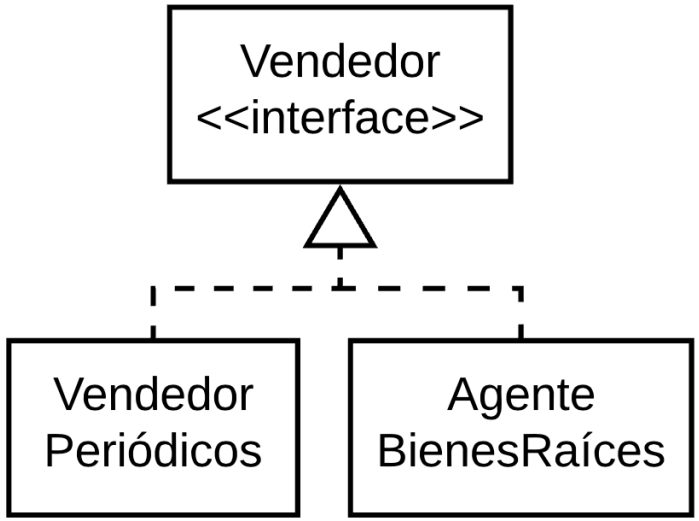
Ejemplo: vendedores que ofrecen y realizan ventas

Una de las principales razones para utilizar interfaces es la **reutilización de código**, la cual va estrechamente ligada al **polimorfismo**.

```
public interface Vendedor {
    String ofrecerVenta();

    void ejecutarVenta(int unValor);

    String consultarInfo();
}
```



Se necesitan programar ventas de periódicos e inmuebles (casas, aptos, etc.). Es necesario hacer **dos clases separadas**, porque actúan de manera distinta: el agente de bienes raíces gana una comisión que es un porcentaje del valor del inmueble, mientras que el vendedor de periódicos no.

4. Ejemplo: tipos de vendedores

Podemos **abstraer** lo que hace un vendedor en una interfaz. ¿Qué tiene que hacer alguien para poder considerarse un vendedor? Bueno, poseer los métodos: ofrecerVenta, ejecutarVenta y consultarInfo.

Entonces se pueden crear funciones que utilicen variables de tipo Vendedor, con lo cual **aseguramos** de que se pueden llamar dichos métodos.

```
static void realizarVenta(Vendedor unVendedor) {
    Console.WriteLine(unVendedor.ofrecerVenta());

    Console.Write("Monto de la venta: ");
    int unValor = Convert.ToInt32(Console.ReadLine());
    unVendedor.ejecutarVenta(unValor);

    Console.WriteLine(unVendedor.consultarInfo());
}
```

Código fuente

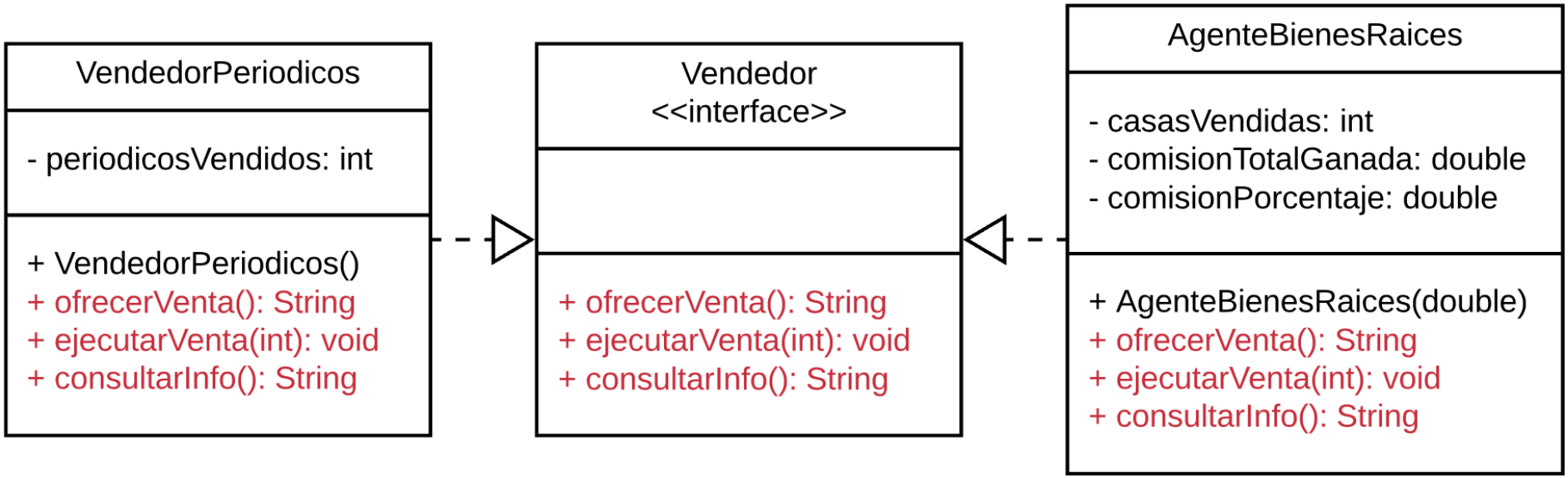
4. Ejemplo: tipos de vendedores

Reutilización de código

Dicho método podrá utilizar objetos de cualquiera de las siguientes clases:

- VendedorPeriódicos.
- AgenteBienesRaíces.
- Cualquier otra clase que implemente la interfaz Vendedor.

¿A qué se parece eso? ¿Cualquier clase que cumpla ciertas restricciones? Sí, a eso se le llama **polimorfismo**.



5. Implementando múltiples interfaces

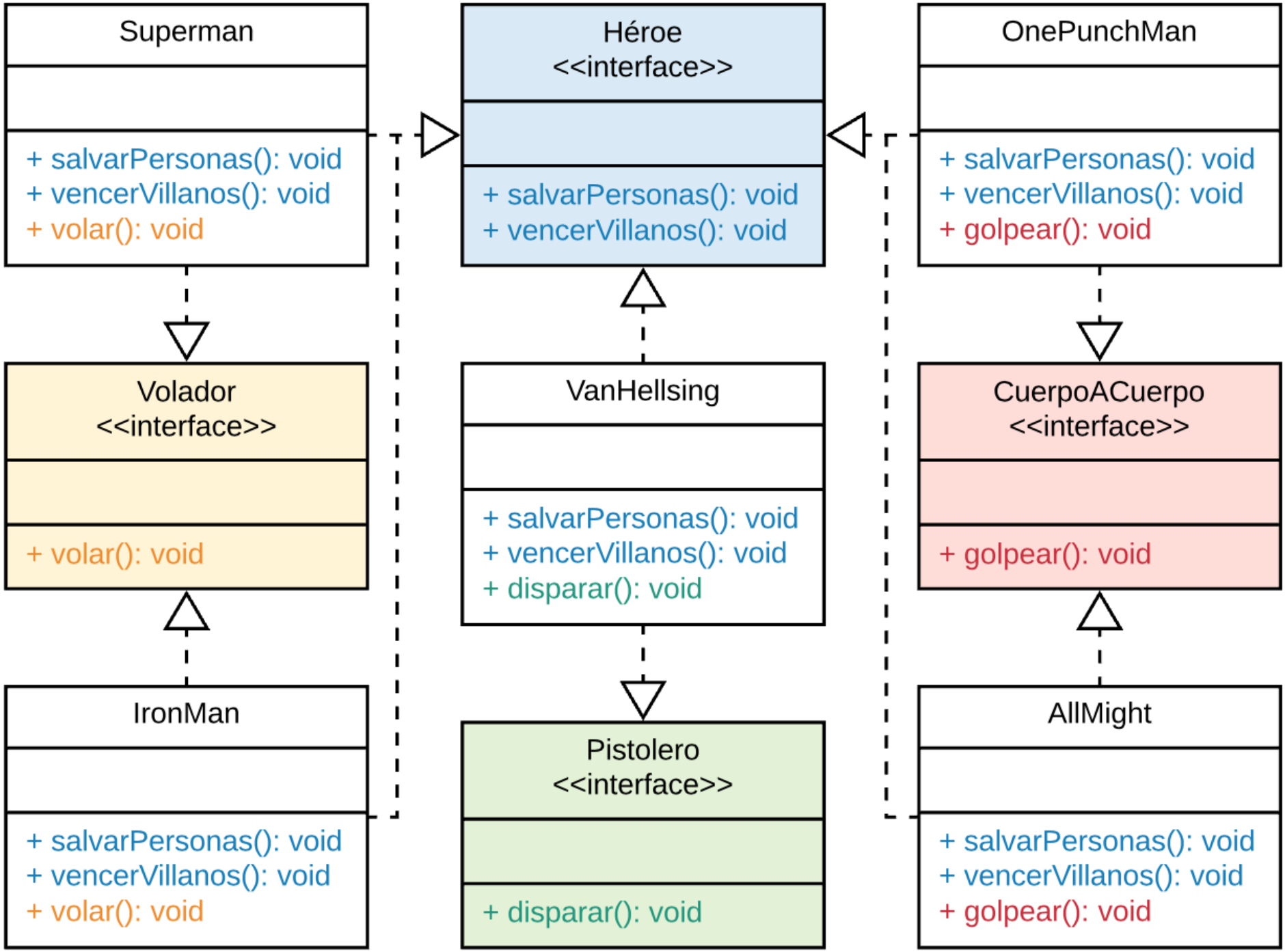
Bueno, se sigue pareciendo a la **herencia** ¿hay algo que las interfaces puedan hacer y que la herencia no?

Una clase puede implementar **múltiples** interfaces. En contraste, una clase puede heredar solamente de una única clase padre.

Esto permite poder crear complejas relaciones que no sería posible implementar solamente con herencia.

- Superman: Héroe y Volador.
- Ironman: Héroe y Volador.
- Van Hellsing: Héroe y Pistolero.
- One Punch Man: Héroe y Cuerpo a cuerpo.
- All Might: Héroe y Cuerpo a cuerpo.

Código Fuente


Superman
+ salvarPersonas(): void
+ vencerVillanos(): void
+ volar(): void
Héroe
<<interface>>
+ salvarPersonas(): void
+ vencerVillanos(): void
OnePunchMan
+ salvarPersonas(): void
+ vencerVillanos(): void
+ golpear(): void
Volador
<<interface>>
+ volar(): void
VanHellsing
+ salvarPersonas(): void
+ vencerVillanos(): void
+ disparar(): void
CuerpoACuerpo
<<interface>>
+ golpear(): void
IronMan
+ salvarPersonas(): void
+ vencerVillanos(): void
+ volar(): void
Pistolero
<<interface>>
+ disparar(): void
AllMight
+ salvarPersonas(): void
+ vencerVillanos(): void
+ golpear(): void

5. Implementando múltiples interfaces

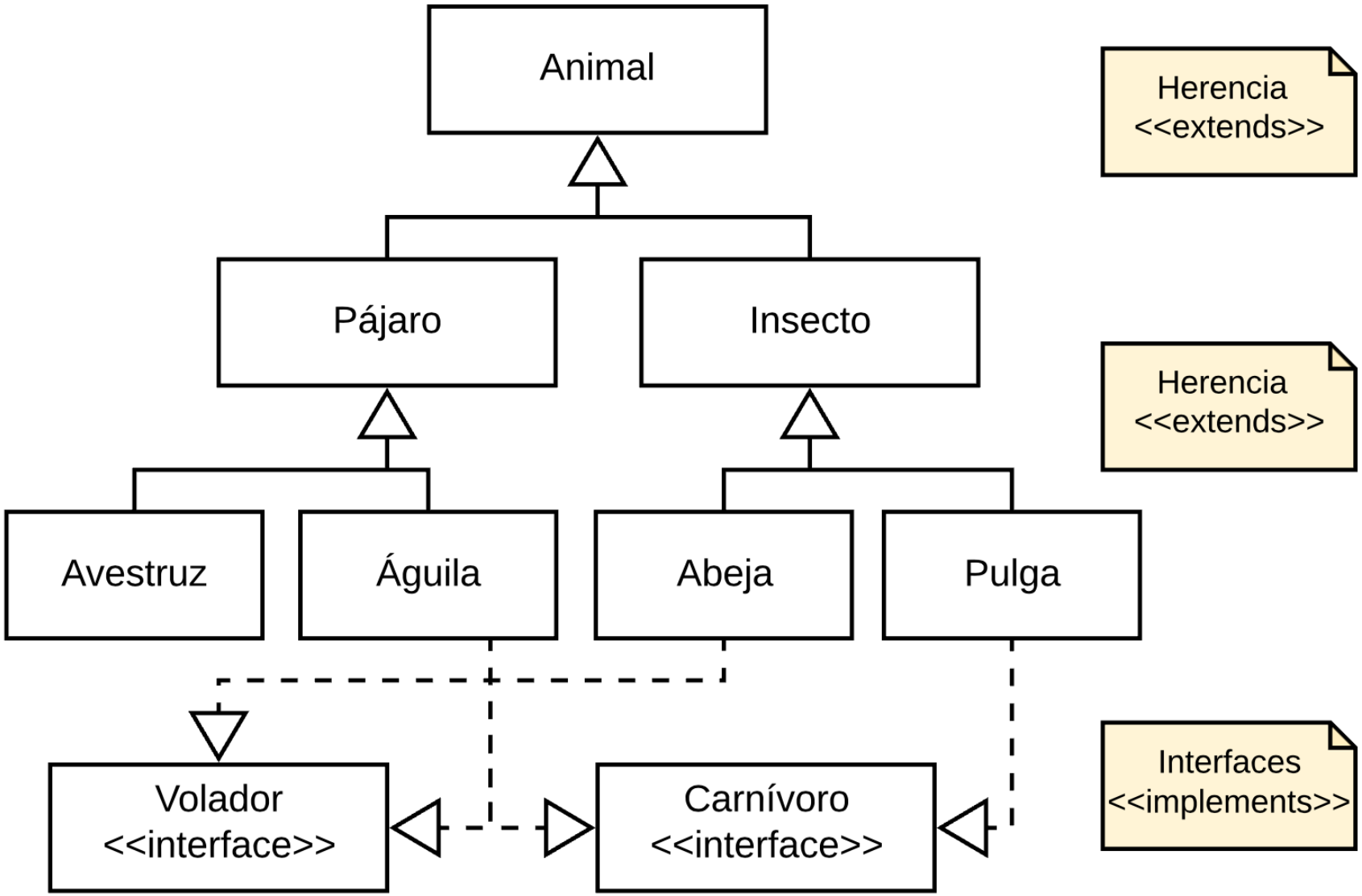
Usar herencia no impide usar interfaces, ni viceversa

También es posible, y se da en la práctica, la mezcla de herencia con interfaces. Imaginemos cómo organizar las siguientes relaciones:

- **Animal:** súper clase, clase abstracta.
- **Pájaro e Insecto:** heredan de Animal, podrían ser abstractas.
- **Avestruz y Águila:** heredan de Pájaro (no abstractas).
  - Avestruz: Ni vuela ni come carne.
  - Águila: Cumple con ser tanto Volador como Carnívoro.
- **Abeja y Pulga:** heredan de Insecto (no abstractas).
  - Abeja: Cumple con ser Volador.
  - Pulga: Cumple con ser Carnívora.

Usted... ¿cómo lo organizaría e implementaría?

5. Implementando múltiples interfaces



Unidad II: Programación orientada a objetos básica

Lic. Ronaldo Armando Canizales Turcios

Departamento de Electrónica e Informática, UCA

Ciclo virtual 01-2021